L7: Entry 2 of 4 File: USPT Mar 7, 2006

DOCUMENT-IDENTIFIER: US 7010523 B2
TITLE: System and method for online analytical processing

PRIOR-PUBLICATION:

DOC-ID	DATE
US 20040133552 A1	July 8, 2004

Abstract Text (1):
A system and method for analyzing data is described, in which an application programming interface (API) is provided to permit an online analytical processing (OLAP) application to manipulate data and queries in a model close to the business model the OLAP application was designed to support. A data server is provided to translate between the object-oriented representation and the native database query format. In one embodiment, a multidimensional virtual cursor is implemented to further simplify the logic of the OLAP application.

Brief Summary Text (14):
The present invention addresses these and other vital needs by employing several features singly and in combination, including an object-oriented query representation, deployment of an OLAP data server apart from the OLAP application, and a multidimensional virtual cursor.

Brief Summary Text (15):
One aspect of the invention involves the specification and construction of queries in the OLAP application in an object-oriented representation rather than by a textual query such as SQL. The query objects resemble the business model of the OLAP application rather than the relational database model of the data warehouse. The execution of the query objects can be performed by a separate data server for providing the OLAP services, with the OLAP application holding remote references to the query objects. More specifically, the objects represent the query state, and the OLAP application refines the queries by invoking methods on the query objects. These actions cause corresponding methods to be invoked on the data server objects remotely. In this configuration, the data server can easily determine how the query has been altered and thus perform any conversions necessary between the object representation and the relational database's own query representation without encumbering the OLAP application itself.

Description Paragraph (7):
FIG. 5 is a class diagram for a cursor object according to one embodiment of the present invention.

Description Paragraph (20):
A "schema" is a collection of relational database objects. Two types of schemas are characteristic of a data warehouse: a star schema and a snowflake schema. A star schema comprises one or more fact tables related to one or more dimension tables. The relationships are defined through foreign keys and metadata. A snowflake schema is a star schema that has been partially or fully normalized to reduce the number of duplicate values in the dimension tables.

Description Paragraph (26):
Metadata is data that describes the data and objects in the relational database 121 for fetching and computing the data correctly. Generally, metadata can be taken to mean the fact

that a data source exists, as well as the structure and characteristics of the data in that data source. For example, the facts that a unitsSold measure exists, that the unitsSold measure contains numeric values, and that the unitsSold measure is dimensioned by geography and product are considered metadata. By contrast, the fact that 30 widgets were sold in 1998 in Tallahassee, Fla. is considered to be data. Concerning dimension members, the facts that a geography dimension exists and that it contains string values as members are other examples of metadata, but the fact that geography contains the particular string "Tallahassee, Fla." is data. Similarly, the fact that there is a hierarchy called standard defined against geography, and that it contains three levels called city, state, and region, are all considered metadata, but the fact that "Tallahassee, Fla." is a child of "Fla." is considered to be data.

Description Paragraph (27):
Accordingly, metadata is used to inform the OLAP application 101 about the data that is available within the relational database 121 in a manner so that the OLAP application 101 can define multidimensional objects for analysis. When the OLAP application 101 runs, the OLAP application 101 instantiates these multidimensional objects and populates them with data fetched from the database.

Description Paragraph (30):
Dimensions identify and categorize the OLAP application's data. In a relational database system, dimension members are stored in a dimension table. Each column represents a particular level in a hierarchy. In a star schema, the columns are all in the same table; in a snowflake schema, the columns are in separate tables for each level. Because measures are typically multidimensional, a single value in a measure must be qualified by a member of each dimension to be meaningful. For example, a Sales measure might have dimensions for Product, Geographic Area, and Time. A value in the Sales measure (37854) is only meaningful when it is qualified by a product (DVD Player), a geographic area (Pacific Rim), and Time (March 2001). Defining a dimension in the data warehouse creates a database dimension object, in addition to creating metadata. A dimension object contains the details of the parent-child relationship between columns in a dimension table; it does not contain data. The database dimension object is used by the Summary Advisor and query rewrite to optimize the data warehouse. However, on the multidimensional side, a dimension does contain data, such as the names of individual products, geographic areas, and time periods. The OLAP API uses the metadata, dimension objects, and dimension tables to construct its dimensions.

Description Paragraph (35):
The OLAP application 101 interacts with client software belonging to a online analytical processing application program interface (OLAP API) 103, which is responsible for presenting and managing an object oriented interface to the OLAP application 101 in accordance with a source model that is conceptually much closer to the business model of the OLAP application 101 than a typical relational database model. In one implementation, the OLAP API 103 client software is a set of Java packages containing classes that implement the programming interface to an OLAP service. The OLAP application 101 calls the methods on these classes for discovering, querying, processing, and retrieving data.

Description Paragraph (36):
More specifically, the OLAP API 103 is also responsible for initiating and buffering communications with an OLAP service 105 operating as a data server, for example, over a network connection using the hypertext transfer protocol (HTTP). As described in greater detail hereinafter, the functions of the OLAP API include presenting an object-oriented interface to the OLAP application 101, caching metadata describing the data in the relational database 121 in multidimensional terms and portions of result sets returned to the OLAP application 101 for the relational database, and formatting the data in a cross-tabulation form.

Description Paragraph (37):
The OLAP API 103 is the programming interface for OLAP services. When the OLAP application 101 calls methods on OLAP API 103 classes, the OLAP application 101 uses client software of the OLAP API 103 to communicate with the OLAP service 105, which typically resides on a different platform. The OLAP service module 105 and the relational database management system 121 reside on a data server tier, where the data is stored, selected, and manipulated. Specifically, the

OLAP service 105 is a child process of an instance of the relational database system 121, and the communication between the OLAP API 103 client software and the OLAP service 105 is provided through a protocol such as the Common Object Request Broker Architecture (CORBA).

Description Paragraph (38):
The OLAP service 105 is responsible for receiving and processing requests submitted by the OLAP API 103. The OLAP service 105's responsibilities include translating and formulating relational database queries to the relational database system 121 based on query objects supplied by the OLAP API 103, performing any calculations on the retrieved data including totaling and other forms of aggregation, and batching results for transmission back to the OLAP API 103 on a request-by-request basis to present a virtual result set to the OLAP API 103.

Description Paragraph (42):
In accordance with this approach, a query is modeled as an object that represents a data request whose definition can be incrementally modified even after the query has been used to fetch data. For example, a query can be sorted, filtered, and then sorted again. Each time the query is modified, the query changes state.

Description Paragraph (43):
FIG. 2 is a flowchart illustrating the life cycle of an OLAP query in accordance with one embodiment of the present invention. The OLAP application 101 begins by connecting to a metadata provider (step 201) and viewing the metadata (step 203). As described in more detail hereinafter, a metadata provider is responsible for obtaining the metadata from the relational database 121. If the client running the OLAP application 101 wishes to view the real data (tested at step 205), then the OLAP application 101, in response, connects to another object called a data provider (step 207). Upon connection, the client creates objects called sources, which are used by the OLAP API 103 to construct and represent queries. Typically, the client creates many sources, beginning with one that represents the basic metadata objects that are found in the relational database 121 (step 209). For example, the client might create a query by specifying a subset of the values in a basic measure.

Description Paragraph (46):
Conceptually, this behavior fits well with end-user expectations, but there is also another advantage to the server by modeling queries in this way. Typically, each change to a query definition is small, so it generally takes less work to satisfy the second request if the server already knows the result of the first query. By reusing the same query object, the server has this knowledge and can optimize performance. If, instead, the conventional SQL approach is used, then the server has no way of knowing that the second data request is related in any way to the original request and must re-execute the query from scratch.

Description Paragraph (47):
Of course, if there is no relationship between two data requests, there is no point in reusing the same query object. In this case, the client simply closes the cursor (219), and being not finished with the data (decision point 221), loops back to step 211 to create a separate query object, so that there has been one query object per request. To finish operations (from decision point 221), the data provider is closed (step 223), and then the metadata provider is closed (step 225).

Description Paragraph (53):
To satisfy condition (2), the concept of a metadata provider is introduced. A metadata provider is an interface that guarantees the OLAP application 101 access to the metadata required for defining and executing queries. Various metadata providers can be implemented, one for the simple metadata model and others for more sophisticated metadata models. The OLAP application 101 can use a simple metadata provider to get a bare bones view of the metadata. The simple metadata provider allows the OLAP application 101 to get a metadata object by a unique identifier and to see a set of property values for that object. Once discovered, the metadata objects can form part of a data query.

Description Paragraph (54):
A multidimensional metadata provider can be used to define a standard view of multidimensional

data in terms of familiar concepts such as dimensions, measures, and hierarchies. A class hierarchy for one such multidimensional metadata model is illustrated in FIG. 3. Most of the classes implement metadata objects, such as dimensions and measures. The following list introduces the subclasses of an MdmObject 301 class.

Description Paragraph (56):
An application accesses metadata objects by creating an OLAP API 103 metadata provider and using the metadata provider to discover the available metadata objects in the data store 121. The metadata objects in the relational database 121 map directly to multidimensional metadata model objects that are accessible through the metadata provider. The objects typically map as follows: a dimension maps to MdmHierarchy 319 or MdmListDimension 317; a hierarchy maps to MdmHierarchy 319; a level maps to MdmLevel 321; a measure maps to MdmMeasure 313; an attribute maps to MdmAttribute 311; and a measure folder maps to MdmSchema 303.

Description Paragraph (57):
An MdmSchema 303 represents a set of data that is used for navigational purposes. An MdmSchema 303 is a container for MdmMeasure 313, MdmDimension 307, and other MdmSchema 303 objects. An MdmSchema 303 is equivalent to a folder or directory that contains associated items. Despite the similarity in name, it does not correspond to a relational schema of a relational database. Instead, the MdmSchema 303 corresponds to a measure folder, which can include data from several relational schemas and which was created by a database administrator.

Description Paragraph (58):
Data that is accessible through the OLAP API 103 is arranged under a top-level MdmSchema 303, which is referred to as the root MdmSchema 303. Under the root, there are one or more subschemas. To begin navigating the metadata, an application calls the getRootSchema method on the metadata provider. The root MdmSchema 303 contains all the MdmMeasure 313 and MdmDimension 307 objects that are in the relational database 121. That is, if the root MdmSchema 303 has subschemas that contain MdmMeasure 313 and MdmDimension 307 objects, the root MdmSchema 303 also contains those objects. An MdmSchema 303 has methods for getting all the MdmMeasure 313, MdmDimension 307, and MdmSchema 303 objects contained therein. The root MdmSchema 303 also has a method for getting the measure MdmDimension 307, whose elements are all the MdmMeasure 313 objects in the relational database 121.

Description Paragraph (59):
An MdmSource 305 represents a measure, dimension, or other set of data (such as an attribute) that is used for analysis. This abstract class is the basis for some important multidimensional metadata model classes, such as MdmMeasure 313, MdmDimension 307, and MdmAttribute 311. MdmSource 305 objects represent data, but they need not provide the ability to create queries on that data. Their function is informational, recording the existence, structure, and characteristics of the data. They need not give access to the data values. In order to access the data values for a given MdmSource 305, the OLAP application 101 calls the getSource method on the MdmSource 305. This method returns a source through which the OLAP application 101 creates queries on the data represented by the MdmSource 305. A source that is the result of the getSource method on an MdmSource 305 is called a primary source. The OLAP application 101 creates new source objects from this primary source as it selects, calculates, and otherwise manipulates the data. Each new source specifies a new query.

Description Paragraph (62):
The following concrete subclasses of MdmDimension 307 represent the specific kinds of MdmDimension 307 objects that can be used in analysis: MdmLevel 321, MdmHierarchy 319, and MdmListDimension 317.

Description Paragraph (63):
MdmLevel 321 represents a list of elements that supply one level of a hierarchical structure. Each element can have a parent and one or more children. The parents and children of a given MdmLevel 321 element are not within the given MdmLevel 321. They are elements of different MdmLevel 321 objects.

Description Paragraph (64):

MdmHierarchy 319 represents a list of elements arranged in a hierarchical structure that has levels based on parent-child relationships. Each element can have a parent and one or more children, and all of these elements are within the MdmHierarchy 319. Though the parent and child elements are within the MdmHierarchy 319, they correspond to elements in MdmLevel objects. Therefore, loosely speaking, an MdmHierarchy 319 is composed of MdmLevel 321 objects. Some MdmHierarchy 319 objects are simply composed of MdmLevel 321 objects. Others are unions of one or more subordinate MdmHierarchy 319 objects, which in turn, are composed of MdmLevel 321 objects.

Description Paragraph (66):
An MdmDimension 307 can have one or more MdmAttribute 311 objects. Each of these MdmAttribute 311 objects maps the elements of the MdmDimension 307 to values representing some characteristic of the elements. To obtain the MdmAttribute 311 objects for a given MdmDimension 307, its getAttributes method may be invoked.

Description Paragraph (67):
An MdmDimension 307 has an MdmDimensionDefinition (not shown), which represents the structure of the underlying data, and an MdmDimensionMemberType (not shown), which represents the basic nature of the elements. These two objects hold important information about the MdmDimension to which they belong. For a given MdmDimension 307, the getDefinition and getMemberType methods are used to obtain these related objects.

Description Paragraph (70):
An MdmDimensionMemberType indicates the basic nature of the elements in the MdmDimension 307. It holds a description for each element, and it often provides methods for finding out other information about individual elements. The MdmDimensionMemberType class is abstract. Therefore, instances are always one of the following subclasses: (1) MdmTimeMemberType, which indicates that the MdmDimension 307 elements represent time periods (an MdmTimeMemberType has methods for finding out the end date and time span for each element); MdmMeasureMemberType, which indicates that the MdmDimension 307 elements are all the MdmMeasure objects in the data store. There is only one MdmDimension 307 with an MdmMeasureMemberType, and it is referred to as the measure MdmDimension 307 (the measure MdmDimension 307 can be obtained by calling the getMeasureDimension method on the root MdmSchema); and MdmStandardMemberType, which indicates that the MdmDimension 307 elements have no specific characteristics. Most MdmDimension 307 objects have an MdmStandardMemberType.

Description Paragraph (71):
An MdmLevel 321 is an MdmHierarchicalDimension 315 whose parents and children are elements from other MdmLevel 321 objects. The elements from a given MdmLevel 321 correspond to a subset of the elements in an MdmHierarchy 319. A given MdmLevel 321 is typically based on a level that was specified a column in a database table to provide the elements for the level. Even though the elements of an MdmLevel 321 have parent-child relationships, an MdmLevel 321 is represented as a simple list. The parent-child relationships among the elements are recorded in the parent and ancestors attributes, which can be obtained by calling the getParentRelation and getAncestorsRelation methods on the MdmLevel 321. Sometimes the parent and ancestors attributes are referred to as parent and ancestors relations. Typically, an MdmLevel 321 has an MdmBaseDimensionDefinition, because the underlying data is structured as a single list.

Description Paragraph (75):
A level MdmHierarchy represents a hierarchical structure whose regions are MdmLevel 321 objects. For example, a level MdmHierarchy for calendar year might have as its regions MdmLevel 321 objects for year, quarter, month and day. A level MdmHierarchy has an MdmUnionDimensionDefinition, and its regions are MdmLevel 321 objects. The return value from its getHierarchyType method is LEVEL_HIERARCHY. A level MdmHierarchy is based on a hierarchy that was defined by a database administrator in the relational database 121.

Description Paragraph (76):
A union MdmHierarchy represents a dimension that has one or more subordinate hierarchical structures. These structures are represented by one or more level MdmHierarchy 319 objects. An example of an MdmHierarchy 319 with two structures is a union MdmHierarchy for time that has

two regions, one for the calendar year and another for the fiscal year. Each region is a level MdmHierarchy. A union MdmHierarchy has an MdmUnionDimensionDefinition and its regions are MdmHierarchy 319 objects. The return value from its getHierarchyType method is UNION_HIERARCHY. A union MdmHierarchy is based on a dimension that was defined as having one or more hierarchies in the relational database 121.

Description Paragraph (78):
An MdmMeasure 313 represents a set of data that is organized by one or more MdmDimension 307 objects. The structure of the data is similar to that of a multidimensional array. Like the dimensions of an array, the MdmDimension 307 objects that organize an MdmMeasure 313 provide the indexes for identifying individual cells. For example, suppose there is an MdmMeasure 313 for sales data, and the data is organized by product, time, customer, and channel (with channel representing the marketing method, such as direct or indirect). This data can be thought of as occupying a four-dimensional array with the product, time, customer and channel dimensions providing the organizational structure. The values of these four dimensions are indexes for identifying each particular cell in the array, which contains a single sales value. A value is specified for each dimension in order to identify a value in the array. In relational terms, the MdmDimension 307 objects constitute a compound (that is, composite) primary key for the MdmMeasure 313. The values of an MdmMeasure 313 are usually numeric, but this is not necessary.

Description Paragraph (80):
MdmMeasure 313 elements are determined by MdmDimension 307 elements. The set of elements that are in an MdmMeasure 313 is determined by the structure of its MdmDimension 307 objects. That is, each element of an MdmMeasure 313 is identified by a unique combination of elements from its MdmDimension 307 objects. Typically, the MdmDimension 307 objects of an MdmMeasure 313 are union MdmHierarchy objects. That is, they have at least one hierarchical structure. It is important to remember that the elements of a union MdmHierarchy include all of the leaves and all of the nodes for all of the level MdmHierarchy objects that represent its regions. Because of this structure, the values of the elements of an MdmMeasure 313 are of two kinds: (1) values from the fact table column (or fact-table calculation) on which the MdmMeasure 313 is based (these values belong to MdmMeasure 313 elements that are identified by a combination of leaf MdmHierarchy 319 elements) or (2) aggregated values that OLAP services 105 has provided (these values belong to MdmMeasure 313 elements that are identified by at least one node element from an MdmHierarchy 319). The method for aggregation (for example, addition) was specified in the relational database 121.

Description Paragraph (81):
An MdmAttribute 311 represents a particular characteristic of the elements of an MdmDimension 307. An MdmAttribute 311 maps one element of the MdmDimension 307 to a particular value. A typical example is an MdmAttribute 311 that records the gender of each customer in an MdmDimension 307 called mdmCustomersDim. In this case, the elements of the MdmAttribute 311 have the values "Female" and "Male". The values of an MdmAttribute 311 might be String values (such as "Female"), numeric values (such as 45), or objects (such as MdmLevel 321 objects). Like an MdmMeasure 313, an MdmAttribute 311 has elements that are organized by its MdmDimension 307. For example, the gender MdmAttribute has one element (with "Female" or "Male" as its value) for each element of the MdmDimension 307 called mdmCustomersDim.

Description Paragraph (82):
Typically, not all of the elements of an MdmDimension 307 have meaningful mappings to the values of a given MdmAttribute 311. For example, the gender MdmAttribute applies only to the lowest level of mdmCustomersDim, because gender makes no sense for higher levels such as cities or states. If an MdmAttribute 311 does not apply to some elements of an MdmDimension 307, then their MdmAttribute 311 values are null. Some MdmAttribute 311 objects provide a mapping that is one-to-many, rather than one-to-one. Therefore, a given element in an MdmDimension 307 might map to a whole set of MdmAttribute 311 elements. For example, the MdmAttribute 311 that serves as the ancestors attribute for an MdmHierarchy 319 maps each MdmHierarchy 319 element to its set of ancestor MdmHierarchy 319 elements.

Description Paragraph (84):

In one aspect of the invention, queries are represented by two objects: source objects and cursor objects. The specification for a query is represented by a source object. Source objects are not actual result sets but merely describe the data to be retrieved. The result set of a query is a cursor object. Cursor objects are the objects that are used to actually retrieve data from the database.

Description Paragraph (85):
Source objects are immutable. A source object cannot be changed once it has been created. When it is desirable to present a source object as changeable to the OLAP application 101 (for example, to support what-if analysis), a template object is provided to define the source object. As described in greater detail herein below, template objects themselves have state and can be modified at any time.

Description Paragraph (86):
A source class has different subclasses for different data types. Each of the subclasses defines methods that are type-specific versions of various source methods and methods that perform type-specific operations. For example, a "BooleanSource" contains Boolean values, and a "DateSource" contains date objects.

Description Paragraph (87):
In one embodiment, the OLAP API 103 supports the following kinds of source 401 objects: (1) a primary source 403, which corresponds to and has a structure similar to a metadata object from which the primary source 403 created; (2) a fundamental source 405 object, which represents data types and functions that are intrinsic to the OLAP API 103; and (3) a derived source 407, which is created by manipulating existing source objects. Constant, list, and range source objects are simple, nondimensional source objects that can be used as operands when making selections and calculations. Since a source is an object, an object reference to the source must be obtained in order to use the source, and the way in which the object reference to a source is obtained varies by the kind of source.

Description Paragraph (88):
A primary source 403 object is created via a getSource method on a metadata object. A primary source 403 that created from an MdmDimension 307 is a specification for a simple list of elements. This kind of source does not have any keys itself but usually acts as a key to other source objects. A primary source 403 created from an MdmDimension 307 is called a nondimensional source. It can be thought of as a table with only a single column that holds the values of its elements.

Description Paragraph (89):
A primary source 403 created from an MdmMeasure 313 or an MdmAttribute 311 is a specification for a data set that has one or more keys. Each of these keys is a primary source 403 that was created from a MdmDimension 307. In other words, this kind of source represents a set of data that is organized by one or more primary source 403 objects that have been created from MdmDimension objects 307. A primary source 403 created from an MdmMeasure 313 or an MdmAttribute 311 can be conceptualized as a multidimensional array. The source objects that were created from MdmDimension 307 objects and that act as its keys are the dimensions of the array. The values of its dimensions are indexes for identifying each particular cell in the array, which contains a single value. In order to identify a value in the array, a value for each dimension must be specified. Thus, the set of elements that are in a dimensional source is determined by the structure of the source objects that act as its keys.

Description Paragraph (90):
A source that is created from an MdmMeasure 313 or an MdmAttribute 311 can also be conceptualized in relational terms as a table that has one column for its elements and one column for the elements of each of the source objects that act as its keys. A source object that is a key to another source is often a primary key in a table in the underlying database. Consequently, when one source is a key to another source, the source that is the key can be thought of as a foreign key. When a source has foreign keys, the primary key of the source is a composite key (or multisegmented key) that comprises its foreign keys. Each element of one source is identified by a set of elements of the source objects that are its foreign keys.

Description Paragraph (91):
The source objects that act as the keys of a dimensional primary source are known as "inputs." An input is a foreign key to a source object for which values have not yet been specified. A source object that has an input knows the identity and characteristics of the input source but does not know the values of the elements of the input. As a result, when a source has inputs, the primary keys to its elements are not fully specified and the OLAP service 105 cannot identify the elements of the source. Thus, a query specification represented by a source that still has an input is incomplete. Consequently, a cursor cannot be created on a primary source and, therefore, its values cannot be retrieved into the OLAP application 101. To retrieve the values represented by a dimensional primary source, a new source must be derived from it by specifying elements for the values of the source objects that act as its keys as described hereinafter.

Description Paragraph (93):
New source objects 407 can be created from existing source objects by using the methods in the source class and its subclasses or by using the generateSource method in the template class. Template objects are features of the OLAP API 103 that represent end-user concepts such as cubes, edges, and selections. Template objects form a bridge between the requirements of the user interface and the powerful, but abstract, OLAP API 103 logical model. Unlike other OLAP API 103 objects, template objects have mutable state. Consequently, template objects can be modified at any time, even after having been incorporated into some larger source. The source 409 defined by a template is dynamic in the sense that it can be changed. More information about templates is described hereinafter.

Description Paragraph (94):
In one embodiment, the OLAP API 103 includes primitive methods and shortcut methods for deriving new source 413 and 415 objects, respectively. The primitive "join" method is perhaps the single most important source creation method in the OLAP API 103. The primitive join method combines the elements of the "this" source (sometimes called the "base source") and another source (called the "joined source") and filters this result set using a third source (called the "comparison source") in the specified manner. Using an optional parameter, the primitive join method can be used to add the joined source as a dimension (or key) to the new source. Implementations of the OLAP API 103 may provide various shortcut and convenience methods that can be used instead of the primitive join method.

Description Paragraph (95):
Other primitive methods include: "alias" for creating a new source object that is the same as the base source object, but that has the base source as its type; "distinct" for removes the duplicate rows (tuples) in this source object; "extract" for creating a new source that has the base source as an extraction input when the elements of the base Source are other Source objects; "position" for creating a new source object with the same structure as the base source and whose elements are the position of the elements of the base source; and "value" for creating a new source object that has the elements of the base source and that has the base source as an input.

Description Paragraph (96):
The signature of the primitive join method is as follows: "Source join(Source joined, Source comparison, int comparisonRule, boolean visible)" where "joined" is the source that is to be joined to the base source, "comparison" is the source to be used as a filter for the join, "comparisonRule" is the rule that determines how the method uses the comparison source to filter the result set, and "visible" is a flag that specifies whether the joined source object is to be an output of the new source.

Description Paragraph (99):
The result of the join method is a new source object 415. Depending on the complexity of the source objects that are being joined, the resulting source object may be simple or complex. When two nondimensional source objects are joined, the new source is dimensioned by the joined source and the new source is simply the cross-product of the two source objects. When dimensional Source objects are joined, the new source 415 has the combined dimensionality of

the base, joined, and comparison source objects. Additionally, true is specified for the value of the visible parameter, the joined source becomes a dimension or key of the new source.

Description Paragraph (103):
Even though it helps to think of a source object as a tabular or dimensional result set, a source actually is not a result set. Instead, a source object is a specification for a query that defines a result set. As part of this specification, a source object keeps track of the keys for which values have been specified. Looking at keys from this point of view, a source object is said to have two different types of keys: inputs and outputs.

Description Paragraph (104):
Inputs are keys for which values have not yet been specified. When a primary source object has other source objects that act as its keys, these source objects are always inputs. Thus, the query specification represented by a dimensioned primary source or any other source that has an input is incomplete. A cursor for this type of source cannot be created and, consequently, the query specified by the Source cannot be retrieved.

Description Paragraph (112):
To create a cursor on a Source object, all of the keys of the source must be outputs. Consequently, to display a primary dimensional source, values for the keys of that source must first be specified. Specifying values for the keys of a source is called changing inputs to outputs. The need to specify values for the keys of a dimensional source with inputs is so universal, that the OLAP API 103 has a join shortcut method to support it. To specify values for the keys of a dimensional source (thereby changing an input to an output), the following join method can be used, in which the original source is the source object that has the input to become an output and the joined source is the input you want to change: "join (Source joined)". This is a shortcut for the following join method: "join (joined, emptySource, Source. COMPARISON_RULE_REMOVE, true);". The comparison source is the empty source, which has no elements. Consequently, even though the COMPARISON_RULE_REMOVE constant is specified, no elements are removed as a result of the comparison. Also, because the visible flag is set to true, the joined source becomes an output of the new source. Additionally, since many of the methods of source class and its subclasses are actually shortcut and convenience methods that implicitly call the join method, some of these methods also change inputs to outputs.

Description Paragraph (116):
For example, let there be a primary source named unitCost that was created from a MdmMeasure object named mdmUnitCost. The source named unitCost has inputs of timesDim and productsDim, and no outputs. The timesDim and productsDim source objects do not have any inputs or outputs. The order in which the inputs of unitCost are turned into outputs determines the structure of a source on which you can create a cursor.

Description Paragraph (123):
A query is an OLAP API 103 source that specifies the data to be retrieved from the OLAP service 105 and any calculations the OLAP service 105 is to perform on that data. A cursor is the object that retrieves, or "fetches," the result set specified by a source.

Description Paragraph (126):
A Compound Cursor 505 is created for a source that has more than one set of values, which is a Source that has one or more outputs. Each set of values of the source is represented by a child Value Cursor 503 of the Compound Cursor 505. A Compound Cursor has methods for getting its child cursor objects.

Description Paragraph (127):
The structure of a source determines the corresponding structure of the cursor. A source can have nested outputs, which occurs when one or more of the outputs of the source is itself a source with outputs. If a source has a nested output, then the Compound Cursor 505 for that Source has a child Compound Cursor 505 for that nested output. The Compound Cursor 505 coordinates the positions of its child Cursor objects. The current position of the Compound Cursor 505 specifies one set of positions of its child Cursor objects.

Description Paragraph (132):
The current value of any value cursor 503 can be obtained by calling either the getCurrentValue method, which returns an object, or one of the typed variants (not shown), such as getCurrentString or getCurrentDouble. These variants are more efficient than getCurrentValue, but they will fail if the current value is not of the specified type.

Description Paragraph (138):
The amount of data that a Cursor retrieves in a single fetch operation is determined by a fetch size specified for the Cursor. For a Compound Cursor, the amount of data fetched in a single operation is the product of the fetch sizes of all of its descendent ValueCursor objects. The total set of values retrieved in a single fetch is the fetch block for the Cursor. The fetch sizes are determined in order to limit the amount of data the application needs to cache on the local computer and to maximize the efficiency of the fetch by customizing it to meet the needs of the OLTP application's method of displaying of the data.

Description Paragraph (139):
A cursor has a local fetch size if the size of the fetch block is specified for that cursor. Not all of the cursor objects in a Compound Cursor can have local fetch sizes. The structure of a Compound Cursor is like a tree, with the hierarchy of cursor objects starting at the topmost (root) cursor and going down through all the child Cursor objects. Any path through the hierarchy, starting from the root and going down to a leaf Value Cursor, can contain one, and only one, cursor with a local fetch size. Specifying the fetch size on a parent cursor affects all of the child Cursor objects of that parent. This means that a fetch block can contain no more than the number of elements of each child cursor specified by the fetch size.

Description Paragraph (142):
The OLAP application 101 developer should specify fetch sizes on the cursor objects that are used to loop through the result set. For example, for a table view, fetch sizes should be set on the root Cursor, and, for a cross-tab view, fetch sizes should be set on the child cursor objects.

Description Paragraph (145):
The template class is the basis of a very powerful feature of the OLAP API 103. Template objects are to create modifiable source objects. With those source objects, dynamic queries can be incrementally changed in response to end-user selections. Template objects also offer a convenient way to translate user-interface elements into OLAP API 103 operations and objects.

Description Paragraph (146):
The main feature of a template is its ability to produce a dynamic source. That ability is based on two of the other objects that a template uses: instances of the DynamicDefinition and MetadataState classes.

Description Paragraph (148):
The source that a template produces can change because the values, including other source objects, that the template uses to create the source can change. A template stores those values in a MetadataState. A template provides methods to get the current state of the MetadataState, to get or set a value, and to set the state. Those methods are used to change the data values the MetadataState stores.

Description Paragraph (152):
To generate a Source that represents the query that the end user creates in the first dialog box, a template called TopBottomTemplate may be designed. A second template, called SingleSelectionTemplate, may be used to create a source that represents the end user's selections of single values for the dimensions other than the base dimension. The designs of the template objects reflect the user interface elements of the dialog boxes.

Description Paragraph (160):
The source produced by a template can be the result of a series of source operations that create other source objects, such as a series of selections, sorts, calculations, and joins. The code for those operations is placed in the generateSource method of a SourceGenerator for

the template. That method returns the source produced by the template. The operations use the data stored in the MetadataState.

Description Paragraph (161):
An extremely complex query may be built that involves the interactions of dynamic source objects produced by many different template objects. The end result of the query building is a source that defines the entire complex query. If the state of any one of the template objects that is used to create the final source is changed, then the final source represents a result set different than that of the previous source. Consequently, the final query can thereby be modified without having to reproduce all of the operations involved in defining the query.

Description Paragraph (162):
Template objects can be designed to represent elements of the user interface of the OLAP application 101. The template objects turn the selections that the end user makes into OLAP API 103 query-building operations that produce a source. A cursor is then created to fetch the result set defined by the source from the OLAP service 105. Values from the cursor are fetched and displayed to the end user. When an end user makes changes to the selections, the state of the template is changed accordingly. Then the source produced by the template is obtained, a new cursor is created, and the new values are fetched and displayed.

Description Paragraph (164):
The OLAP API 103 is transactional. Each step in creating a query occurs in the context of a transaction. One of the first actions of an OLAP application 101 is to create a Transaction Provider. The Transaction Provider provides transaction objects to the application.

Description Paragraph (165):
The Transaction Provider ensures the following: (1) A transaction is isolated from other Transaction objects. Operations performed in a Transaction are not visible in, and do not affect, other Transaction objects. (2) If an operation in a transaction fails, its effects are undone (the transaction is rolled back). (3) The effects of a completed transaction persist.

Description Paragraph (168):
The OLAP API 103 has the following two types of transaction objects: a read transaction and a write transaction. Initially, the current transaction is a read transaction. A read transaction is required for creating a cursor to fetch data from an OLAP service 105. A write transaction is required for creating a derived source or for changing the state of a template.

Description Paragraph (169):
In the initial read transaction, if a derived Source is created or if the state of a template object is changed, then a child write transaction is automatically generated. That child Transaction becomes the current transaction.

Description Paragraph (170):
If another derived Source is then created or if the template state is changed again, that operation occurs in the same write transaction. Any number of derived source objects can be created, or any number of template state changes can be made, in that same write transaction. These source objects, or the source produced by the template, can be used to define a complex query.

Description Paragraph (178):
A transaction can be rolled back or undone by a rollbackCurrentTransaction method on the Transaction Provider. Rolling back a transaction discards any changes made during that transaction and makes the transaction disappear. After rolling back a transaction, any source objects created or template state changes made in the transaction are no longer valid. Any cursor objects created for those source objects are also invalid. Once a transaction is rolled back, that transaction cannot be prepared and committed. Likewise, once a transaction is committed, it cannot be rolled back.

Other Reference Publication (2):
Qian et al., Translating Object-Oriented Queries to Relational Queries, Mar. 11, 1994, pp. 1-

18. cited by examiner

Other Reference Publication (3):
Buzydlowski et al., A Framework for Object-Oriented On-Line Analytic Processing, ACM 1999, pp. 10-15. cited by examiner

Other Reference Publication (4):
Jan W. Buzydlowski et al. A Framework for Object-Oriented On-Line Analytic Processing, DOLAP '98 Washington DC USA, ACM 1999. cited by other

Other Reference Publication (5):
George Colhat OLAP, Relational, and Multidimensional Database Systems, ACM Sigmod Record, vol. 25, Sep. 1996. cited by other

CLAIMS:

1. A computer-implemented method for analyzing data comprising the steps of: receiving a call from an analytical processing application; constructing a query object based on the call; translating the query object into a textual query for submission to a data warehouse; retrieving data from the data warehouse in response to the submission of the textual query; and providing at least some of the data retrieved from the data warehouse in response to the textual query; wherein said providing at least some of the data retrieved includes: receiving a specification of an extent of a multidimensional cursor from the analytical processing application; and determining said at least some of the data based on the specification.

3. The method of claim 1, further comprising refining the query object by sub-transactions of transactions performed on a database.

4. The method of claim 3, wherein said refining the query object includes: generating a template of a source object based on metadata indicating a description of database data and database objects included in the data warehouse; and modifying the template based on a user input.

5. The method of claim 1, further comprising: receiving a refinement of the query object from the analytical processing application; and initiating a sub-transaction to process a refined query that corresponds to the refinement of the query object.

6. The method of claim 1, wherein said constructing the query object based on the call includes: generating a source object; and generating a cursor object, wherein the source object includes a specification for the query, wherein the cursor object includes a result of the query.

7. The method of claim 1, wherein said translating the query object into the textual query includes: matching an input with an output; and generating a source object including a specification for a result of a query, wherein the input includes a key having an unspecified value, and wherein the output includes a key having a specified value.

10. The method of claim 1, further comprising: selectively rolling back at least one portion of a query associated with the query object.

Previous Doc Next Doc Go to Doc#

Freeform Search

Database: US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Term: 14 and populat$

Display: 50 **Documents in Display Format:** - **Starting with Number** 1

Generate: ○ **Hit List** ◉ **Hit Count** ○ **Side by Side** ○ **Image**

Search Clear Interrupt

Search History

DATE: Tuesday, March 28, 2006 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR			
L7	l4 and populat$	4	L7
L6	l1 and (inflat$ near object$)	1	L6
L5	L4 and inflat$	1	L5
L4	L3 and instantiat$	6	L4
L3	L2 and (object near model)	23	L3
L2	L1 and objects	386	L2
L1	multidimensional near database	569	L1

END OF SEARCH HISTORY